Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Verificación automática del protocolo DP-3T asociado a las aplicaciones COVID-19.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Daniel Martínez Martín

Tutor: Dr. Santiago Escobar Román

Co-tutor: Dr. Julia Sapiña Sanchis

Curso 2020-2021

Resumen

Actualmente, vivimos en una sociedad ultra conectada que gira en torno al intercambio de información entre los dispositivos y usuarios de forma continuada. De la mano de la información va la privacidad y la confidencialidad de los datos, la cual debe de estar preservada y garantizada de la forma óptima y segura posible y en la que cada vez se está imprimiendo un mayor esfuerzo para trabajar en entornos más seguros, con la finalidad de evitar intrusismos o pérdida de datos.

Debido a la situación ocasionada por la pandemia de COVID-19, el Gobierno de España ha puesto a disposición del ciudadano una aplicación capaz de controlar los contactos y contagios de una forma más personal y ágil, realizando un intercambio bidireccional de datos con todos los usuarios que posean la aplicación y que haya habido contacto, ofreciendo la posibilidad de notificar un caso de COVID-19 positivo simplemente pulsando un botón, de forma que se alerte a todos los contactos del usuario de una forma más rápida y efectiva.

En este trabajo, vamos a analizar mediante la herramienta Maude-NPA la seguridad que tiene el protocolo DP-3T; protocolo desarrollado en respuesta a la pandemia que permite el intercambio de identificadores entre contactos a través de Bluetooth Low Energy, sobre el que se basa la APP RadarCovid, realizando una simulación de ejecuciones del protocolo y proponiendo varias situaciones donde un intruso podría obtener información para un uso indebido, y comprobar si existen ejecuciones del protocolo en la que pueda ocurrir alguno de los casos propuestos.

Palabras clave: Bluetooth Low Energy, RadarCovid, DP-3T. Maude-NPA, COVID-19

Abstract

Nowadays, we live in a ultra connected society, that spins around data exchange between devices and user continuously. From data comes privacy and confidentiality, which should be preserved and guaranteed in the best and safest way possible and in which more and more effort is being made to work in safer environments, aiming to avoid data loss and intrusion.

Due to the situation by the pandemic Covid 19, the Gobierno de España has given the citizen anapp able to control the contacts and contagion in a more agile and personal way, doing a bidirectional data exchange with all the users that has the application and have had contact, giving the possibility to notify a positive Covid 19 case with only pressing a button, alerting all the users that had been in contact in a quick and effective way.

In this project, we are going to analyze through the Maude-NPA tool the DP-3T protocol; developed in response to the pandemic situation that allows id through Low Energy Bluetooth, in which the APP Radar Covid is based, doing an execution simulation of the protocol and proposing various situations where an intruder could get information for an undue use, and check if there is any execution of the protocol that can match the proposed cases.

Keywords: Low Energy Bluetooth**,** RadarCovid, DP-3T. Maude-NPA, COVID-19

Agradecimientos

Recién salido de un año que ha sido como mínimo esperpéntico y adaptándome todavía a la nueva normalidad, puedo confirmar que no ha sido uno de los mejores años de mi vida. Sin duda, este último año queda marcado por muchas experiencias, tanto buenas como malas. Tras estar aislado pudiendo hacer poco más que teletrabajar y mirar por la ventana, creo que hablo en general cuando digo que hemos aprendido a valorar todas esas cosas que antes no eran tan importantes. Desde luego que a partir de ahora voy a valorar más a mis amigos y seres queridos, a viajar y a vivir con más intensidad que antes.

Primero de todo, me gustaría agradecer a mi pareja, mis padres y mi hermano, sin cuyo apoyo incondicional y motivación este camino habría sido muy difícil de recorrer. A pesar de la situación que nos ha tocado vivir, siempre me han empujado a hacer lo que me gusta y me han estirado de las orejas cuando me he desviado del camino.

También me gustaría agradecer su ayuda y sus consejos a mi amigo Pedro Salguero, el cual ha sido una gran ayuda, tanto psicológica como académica. Peter sin ti no habría llegado hasta aquí.

Para mi otro amigo Oscar Pellicer tampoco puedo decir otra cosa que no sea gracias. Tu enfoque de las cosas ha sido revitalizante y puedo decir que con tu perspectiva he sido capaz de ver cosas que antes no estaban tan claras.

Por último, pero no menos importante, agradecer a ni tutor, Santiago Escobar, por su gran paciencia y su adaptabilidad a mis horarios, que no siempre han sido los más cómodos para él pero siempre me ha ayudado con una sonrisa.

Tabla de contenidos

Tabla de figuras

1. Introducción

La pandemia generada por COVID-19 ha tenido severas repercusiones sobre la sociedad, especialmente sobre el sistema sanitario, ya que como es de conocimiento público, ha estado saturado en varias ocasiones debido a la gran cantidad de casos positivos que ha habido a lo largo de las diferentes olas de contagio.

Junto a esto, apareció la figura del rastreador, que es aquella persona cuya función consiste en rastrear todos los contactos que ha tenido una persona en los días previos a la manifestación de síntomas del virus o de dar positivo en la prueba en caso de ser asintomático. Viendo esto, es fácil deducir que debido al gran número de casos se trata de una ardua tarea a realizar, para la que se requiere un gran número de personal realizando estas funciones si lo que deseamos es que como mínimo este proceso sea funcional.

Para atajar los problemas que esta situación pueda ocasionar, reducir costes y aumentar el porcentaje de éxito o veracidad de este trabajo, desde la OMS se ha reconocido la utilidad de aplicaciones de rastreo automático como es la aplicación RadarCovid, la cual mediante la creación de un ID efímero y su envío a los usuarios cercanos, permite crear una cadena de estos; que en caso de que uno de los miembros de la cadena sea positivo, con indicarlo en la aplicación el resto de los miembros de la cadena van a ser notificados y por supuesto, el propio Gobierno de España. Cabe destacar, que los datos indican que el uso de este sistema ha sido capaz de detectar hasta el doble de positivos que de la forma tradicional y sin reportar ningún falso positivo, según indica el propio Gobierno de España a través de la web radarcovid.org.

Esta aplicación se encarga de compartir este ID efímero con los usuarios cercanos a través de la tecnología Low Energy Bluetooth, de forma automática con todos los usuarios que tienen esta misma aplicación instalada en su Smartphone. A priori, parece una solución bastante sencilla, pero eso no es todo, necesitamos que la transferencia de esta información sea lo más segura e invulnerable que podamos, ya que al final estamos compartiendo información privada de cada usuario.

Por esto mismo, la aplicación RadarCovid, se basa en el protocolo *DP-3T*, es decir, *Decentralized Privacy-Preserving Proximity Tracing* o también conocido como protocolo de Rastreo de Proximidad Descentralizado para Preservar la Privacidad, protocolo nacido a raíz de la pandemia mundial para tratar de preservar de la mejor manera posible la privacidad de cada usuario.

En este trabajo, vamos a tratar de determinar como de efectivo es este protocolo, simulando su funcionamiento en la herramienta de análisis *Maude-NPA* y obteniendo una serie de trazas asociadas a situaciones donde un intruso podría ser capaz de vulnerar la privacidad de los usuarios. De esta forma seremos capaces de valorar si el protocolo es lo suficientemente robusto y seguro.

1.1 Justificación

Hay dos motivos principales que me han motivado a realizar este trabajo de final de grado (TFG). El primero de ellos es la seguridad de los datos que compartimos con aplicaciones y usuarios. Cada día, todas las personas estamos compartiendo información constantemente desde el momento en el que usamos cualquiera de las aplicaciones que tenemos instaladas en nuestros smartphones u ordenadores. El hacking, la obtención de datos de forma fraudulenta y el uso malintencionado de los mismos se encuentra en auge. Los esfuerzos tanto de la comunidad criptográfica como de los expertos de seguridad son cada vez mayores, las medidas de seguridad cada vez son más robustas y los usuarios a veces no son todo lo conscientes del valor de su privacidad.

Por todo lo mentado anteriormente, me parece muy interesante analizar el protocolo *DP-3T*, ya que ha sido desarrollado por una entidad pública en respuesta a una situación de emergencia para comprobar como de seguros están nuestros datos y también poder vislumbrar el gran esfuerzo que se ha hecho en tan poco tiempo para desarrollar un protocolo lo más seguro posible.

Por otro lado, sin duda lo que hemos vivido estos meses y sin duda lo que aún nos queda por delante, ha sido una situación, como mínimo especial. La sensibilidad que ha creado en todos nosotros ha sido algo único y me ha parecido más que interesante estudiar una herramienta y un protocolo que han nacido con el único objetivo de reducir los contagios y la mortalidad de este virus en la mayor medida posible.

1.2 Objetivos

Los objetivos que hemos planteado en este trabajo con carácter general son:

- Estudio y planteamiento del funcionamiento del protocolo *DP-3T*.
- Modelado del protocolo en la herramienta *Maude-NPA*.
- Extraer posibles casos de vulneración.
- Modelado de los posibles ataques en la herramienta.
- Obtención de resultados y análisis de estos.

1.3 Estructura

La estructura que hemos planteado para la realización del trabajo es la siguiente: primero vamos a tratar en el capítulo dos el estado del arte del protocolo *DP-3T* con respecto al resto de protocolos de envió por proximidad, teniendo en cuenta la mayoría de las tecnologías que han nacido en respuesta a la pandemia y también trataremos el estado del arte de las herramientas de verificación de protocolos. Seguidamente, conoceremos la herramienta *Maude-NPA* en el capítulo tres y el protocolo *DP-3T* en el capítulo cuatro.

A ello continuará el desarrollo del trabajo en los capítulos cinco y seis, que engloba el modelado del protocolo en *Maude-NPA*, la creación de una ejecución regular para confirmar que el protocolo modelado es funcional. También modelaremos varios posibles ataques a la seguridad de este protocolo, analizando los resultados y poniéndolos en común.

Por último, estableceremos las conclusiones y la bibliografía.

2. Estado del arte

Actualmente (2021), existen varios protocolos de rastreo digital de contactos que sirven para el envío de mensajes por proximidad y por supuesto, también hay varias herramientas de verificación. A continuación, vamos a ver el estado del arte de estos dos grupos por separado.

2.1 Rastreo digital de contactos

El rastreo digital de contactos es un tipo de rastreo en el que los contactos de una persona son identificados de forma automática usando los smartphones. La idea del rastreo digital comenzó en 2007, cuando se demostró que era efectivo en los años próximos, pero la investigación fue frenada ya que era necesaria una adopción generalizada.

Durante la crisis sanitaria producida por COVID-19 este método tuvo un mayor auge, tanto empresas públicas como privadas han visto el potencial que tiene una herramienta que es capaz de generar automáticamente un historial de contactos, ganando una mayor veracidad en los datos y generando un aumento de contactos que se indica que es hasta el doble que los conseguidos a través de los rastreadores manuales.

Teniendo en cuenta la limitación dada por los dispositivos móviles, vamos a diferenciar las herramientas de rastreo automáticas en dos bloques: los que rastrean través del GPS y los que lo hacen a través de Bluetooth. Además de la proximidad, los protocolos de rastreo también pueden estar diferenciados entre centralizados y descentralizados, diferenciando quien mantiene el

historial de contactos de cada usuario en cada caso; en el centralizado lo mantiene un servidor central y en el descentralizado lo mantienen los usuarios individuales.

Entre las diferentes metodologías de rastreo digital podemos diferenciar las siguientes subsecciones.[1]

2.1.1 Rastreo de proximidad Bluetooth

Este método utiliza la tecnología Bluetooth Low Energy para rastrear los contactos de los usuarios. Se ha convertido en la base de las aplicaciones de rastreo de contactos, principalmente para comprobar nuestra exposición a la COVID-19. Habitualmente se utiliza para el intercambio de identificadores de forma anónima que van cambiando cada poco tiempo y que son almacenados localmente para cada usuario. [2]

2.1.2 Seguimiento de ubicación

Este método de rastreo utiliza las torres de telefonía móvil o el GPS. Este método tiene la ventaja significativa de que no necesita que el usuario instale una aplicación para realizar el rastreo, a través de la información recopilada por las operadoras de teléfono, es posible saber dónde ha estado un usuario en qué momento y acompañado de qué persona. Este método fue utilizado en Israel, donde sirvió para aumentar el porcentaje de adopción, aunque fuera de manera obligatoria y vulnerando la intimidad de cada uno de los usuarios.

2.1.3 Etiquetado con códigos GEO-QR

El método de etiquetado con códigos GEO-QR consiste en colocar códigos QR en puntos estratégicos. La gente que se desplace a estos puntos debe de leer ese código para quedar registrado, tanto el usuario como la hora en ese lugar concreto.

En caso de que haya algún contacto positivo, solamente deben de comprobar el histórico de usuarios que han registrado ese QR para rastrear los contacto. Este método no necesita la instalación de ninguna aplicación y dan control total de la privacidad a los usuarios.

2.1.4 CCTV con reconocimiento facial

Este método utiliza los CCTV, Circuitos Cerrados de Televisión, para el reconocimiento facial con el fin de detectar casos confirmados y a todos los usuarios que estén infringiendo las normativas sanitarias. Colocando cámaras en zonas determinadas, el software de reconocimiento facial es capaz de generar un histórico de contactos que han transitado esa ubicación.

2.2 Protocolos de rastreo por proximidad

A raíz de los métodos vistos en el apartado anterior, se han desarrollado una serie de protocolos basados todos en *EphID* (IDs efímeros) para el rastreo por proximidad. [3]

2.2.1 Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT)

El *PEPP-PT* es un sistema de rastreo de proximidad con preservación de privacidad, respaldado por la GDPR. Se trata de un protocolo centralizado, donde el histórico de contactos lo mantiene el servidor y es de código cerrado. El protocolo que vamos a analizar, el *DP-3T* está basado en este protocolo, con el que comparte muchas similitudes.

El *PEPP-PT* transmite un ID temporal creado de forma aleatoria que se transmite a otros usuarios a través de la tecnología de Bluetooth Low Energy. El principal problema de este protocolo es que, al ser un protocolo centralizado, el servidor es un foco para los ataques, además, este servidor mantiene los ID temporales, que expone a mayor peligro la información de los usuarios. [*4*]

2.2.2 Notificación de exposición

Se trata de un protocolo descentralizado, donde el histórico de contactos lo mantiene cada usuario y es de código abierto, desarrollado por Google y Apple.

Es muy similar al *DP-3T*, pero este se desarrolló a nivel de sistema operativo.

2.2.3 Decentralized Privacy-Preserved Proximity Tracing (DP-3T)

Este protocolo va a ser el que vamos a analizar en el trabajo, se trata de un protocolo descentralizado, y de código abierto.

Ha sido elegido por el Gobierno de España para la creación de la aplicación RadarCovid, la cual se puso al servicio de la ciudadanía para tratar de llevar el seguimiento de los contactos de todos los usuarios de una forma automática. Por desgracia, no tuvo la aceptación esperada.

En general, el protocolo genera varios identificadores efímeros que están derivados de una clave secreta que se genera de forma diaria. Este identificador es el que se comparte con los usuarios que han sido contacto y el histórico se almacena en los propios dispositivos. El servidor se encarga exclusivamente de reenviar la información que le envían los usuarios.[5]

2.2.4 BlueTrace / OpenTrace

Protocolo centralizado de código abierto. Conocido por haber sufrido una fuga accidental de datos de usuarios.

Cuando un usuario se registra en la aplicación, el servidor le genera un ID aleatorio que va asignado al número de teléfono del usuario. La razón por la que se almacena el número de teléfono es para ponerse en contacto con el usuario en caso de que sea contacto, pero al mantener esta información en el servidor, está poniendo en riesgo esta información.

Cuando un usuario tiene contacto con otro, los usuarios comparten una serie de mensajes, pero en ningún momento comparten identificadores por los que otro usuario les pueda rastrear. Es el servidor el que tiene estos identificadores que genera cada usuario, y en caso de contacto, como tiene la relación entre los identificadores de cada usuario y su teléfono de contacto, puede alertar al usuario.

2.2.5 Protocolo TCN

Protocolo descentralizado de código abierto, desarrollado por *CovidWatch*. Funciona a través de la creación de un número de contacto TCN, que comparte y recibe de otros usuarios.

Cada usuario posee una base de números de contacto aleatorios. En el momento en el que dos usuarios tienen contacto, generan un código temporal único, el cual ambos usuarios almacenan localmente.

En el momento en que un usuario da positivo en COVID-19, tiene que reportarlo al servidor. Cuando se reporta al servidor, se envía un informe al propio servidor, en ese momento, todos los usuarios lo descargan y realizan la búsqueda del código temporal único que podrían compartir

con ese usuario. En caso de que lo encuentre, significará que han estado en contacto y ese cliente lo comunica.

2.3 Herramientas de verificación de protocolos

Para realizar el estudio propuesto para el trabajo, necesitamos usar una herramienta de verificación de protocolos. Estas herramientas sirven para verificar si un protocolo es seguro, modelando primeramente el funcionamiento del protocolo y proponiendo casuísticas en las que puede haber una vulneración.

En los tiempos que vivimos, la cantidad de protocolos que existen y que se siguen desarrollando es muy numerosa, por lo que este tipo de análisis o verificaciones son cada vez más necesarias, todos los desarrolladores van a querer implementar el protocolo más seguro que puedan en las aplicaciones o servicios que están desarrollando.

Debido al crecimiento exponencial que han tenido y siguen teniendo las conexiones a Internet y los intercambios de datos en general, estas herramientas han ido consolidándose cada vez más como herramientas necesarias a la hora de evaluar la seguridad y las vulnerabilidades de los protocolos.

Existen diferentes herramientas capaces de realizar esta función, en el caso particular de este trabajo vamos a usar *Maude-NPA*, una herramienta de verificación automática que nos va a ayudar a comprobar si este protocolo es seguro. La motivación para usar esta herramienta en lugar de otras existentes (como pueden ser *Verifpal*, *Proverif*…) ha sido que se trata de una herramienta mucho más abstracta, es decir, al no tener un lenguaje demasiado rígido, podemos programar funciones algebraicas más complejas.[6]

3. Fundamentos teóricos

Para poder analizar en la herramienta *Maude-NPA* el protocolo DP-3T, primero debemos comprender mejor el protocolo y su funcionamiento.

3.1 DP-3T: Origen y objetivo

Como respuesta a la pandemia COVID-19, el proyecto *Pan-European Privacy-Preserving Proximity Tracing* (*PEPP-PT*) propuso el sistema *Decentralized Privacy-Preserving Proximity Tracing (DP-3T).* Un sistema que no necesita que la localización y los contactos de los usuarios sean almacenados en un servidor central, minimizando los datos que este tiene que almacenar y reduciendo el riesgo de pérdida de datos.

Se creo con la finalidad de alertar a los usuarios que han estado en contacto estrecho con un caso de COVID-19 positivo por una duración prolongada. Aprovechando que actualmente todos los usuarios de una edad cada vez más corta ya llevan un smartphone consigo, es una forma muy eficiente de rastrear los contactos de estos. Por supuesto, esta herramienta está planteada como un complemento, no como un sustituto del rastreo manual. Con este protocolo se busca automatizar las alertas de COVID-19 para que las autoridades sanitarias puedan invertir mayor tiempo y esfuerzo en otras tareas.

Siempre que hablamos de transferencia de datos tenemos que hablar obligatoriamente de privacidad. Teniendo en cuenta que este protocolo es descentralizado y que el servidor no almacena información personal de los usuarios, el protocolo es capaz de proveer a los usuarios de las siguientes protecciones de seguridad y privacidad:

- **Garantiza la minimización de los datos:** El servidor no almacena ninguna información de proximidad de los usuarios. Las autoridades sanitarias no tienen ninguna información del usuario hasta que el propio usuario comunica el contagio.
- **Previene el abuso de datos:** Ya que el servidor central solo recibe la cantidad de datos mínima para funcionar, no puede usar estos datos para otros fines.
- **Previene el rastreo de los usuarios:** Es imposible trazar a un usuario que no ha reportado al servidor que es positivo.
- **Desmantelamiento seguro:** El sistema se desmantelará una vez acabe la pandemia. En ese momento los datos de los usuarios positivos dejarán de subirse al servidor y todos los datos que tiene almacenados se borrarán a los 14 días.

3.2 DP-3T: Funcionamiento

El funcionamiento del DP-3T se basa en la creación de un identificador efímero, conocido como *EphID*, generado a partir de una clave secreta diferente cada día basado en *Secret Key* (SK) o Clave Secreta.

Cada día, todos los usuarios crean aleatoriamente una *SK*, conocida como SK_t, que es generada a partir de la SK del día anterior, siguiendo la ecuación:

$$SK_t = H\ (SK_{t-1})$$

Donde H es una función criptográfica Hash. Es a partir de ese SK_t que se generan los EphID que el usuario va a distribuir a todos los usuarios durante el día.

Para evitar que identifiquen al usuario por su *EphID*, todos los identificadores efímeros cambian continuamente. El tiempo que tarda cada usuario en cambiar el *EphID* se define como *epoch* (época o era). La longitud del *epoch*, en minutos, es un parámetro configurable L.

Al comienzo de cada día, se crean n *EphID*, que son los que van a repartir durante el día:

$$n = \frac{(24 * 60)}{L}$$

Los usuarios eligen de forma aleatoria que *EphID* es difundido durante L minutos. Estos identificadores se transmiten vía Bluetooth Low Energy, los smartphones envían y reciben continuamente estos indicadores, junto con el día en el que lo reciben y el tiempo de exposición. Por cada identificador recibido se almacena el propio *EphID*, la medida de exposición y el día de recepción del identificador. Tanto los identificadores efímeros recibidos como el SK_t son mantenidos por 14 días.

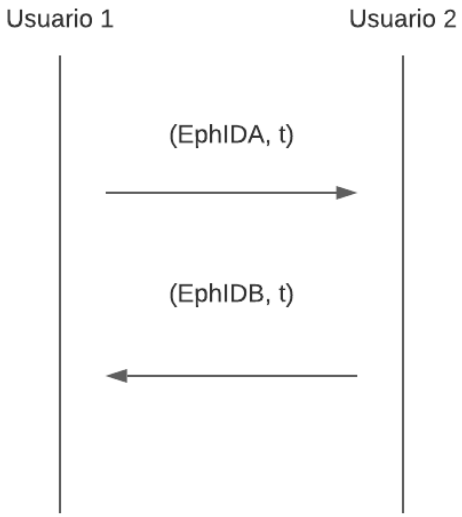

Figura 1. Ejemplo del intercambio del Ephimeral ID entre dos usuarios.

Este proceso de rastreo es soportado por un servidor central, el cual se encarga de almacenar y enviar información anónima a las aplicaciones de todos los dispositivos. El servidor estable una relación de confianza, donde se compromete a no añadir ni eliminar información y a estar siempre disponible. Este servidor tiene la única función de ser una plataforma de comunicación, no se encarga de realizar ningún proceso, por lo que la privacidad de los datos de los usuarios no depende del servidor, ya que no los contienen.

Una vez las autoridades sanitarias han confirmado que un usuario es positivo en COVID-19, el usuario manda al servidor central la semilla SK_t y el día t del primer día que el usuario es determinado como contagioso. Una vez enviado, el usuario elimina ese SK_t.

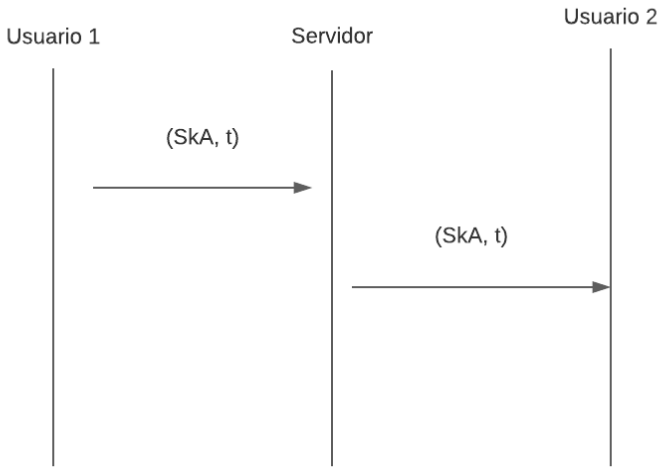

Figura 2. Ejemplo de la comunicación entre el usuario positivo y el servidor, y posterior envío de datos del positivo al resto de usuarios.

En el momento en el que el servidor tiene estos datos, comienza a enviarlos a todos los dispositivos, para que reconstruyan un *EphID* con esos datos que pueda coincidir con algunos de los *EphID* que puedan tener almacenados de los últimos 14 días. Si algún usuario encuentra el *EphID* que ha creado en su histórico de contactos, debe reportar su estado al servidor central y el proceso que acabamos de ver vuelve a comenzar.

Este protocolo ha sido utilizado en el desarrollo de la App RadarCovid, una aplicación lanzada por el Gobierno de España y desde el que han incitado a todos los ciudadanos a instalarla para ayudar a las entidades sanitarias a notificar a todos los contactos de un caso positivo. [7]

4. Material y métodos

En este trabajo, como ya hemos mencionado antes, el material principal para el análisis es una herramienta de análisis criptográfico. En nuestro caso concreto vamos a utilizar *Maude-NRL Protocol Analyzer*; desarrollado por Escobar, Meadows y Meseguer en 2017, en su versión 3.1.4, también conocido como *Maude-NPA*, basado en el lenguaje de programación *Maude*.

4.1 Maude-NPA

Se trata de una herramienta de análisis para protocolos criptográficos que tiene en cuenta propiedades algebraicas que no son compatibles con otras herramientas. *Maude-NPA* está basado en *NRL Protocol Analyzer*, tomando como referencia la unificación y la búsqueda hacia atrás. La principal diferencia con su predecesor es que provee de una función soporte para teorías que impliquen cualquier combinación de asociatividad, conmutatividad y axiomas de identidad. A

esto se suma que al trabajar sobre la unificación ecuacional permite el aumento de estos axiomas con reglas de reescritura, pudiendo alcanzar así la cancelación de cifrado y descifrado, la exposición, etc.

En este trabajo vamos a modelar el protocolo en la herramienta y vamos a proponer varias trazas de ejecución donde un intruso obtiene información de los usuarios. Lo hacemos de esta forma porque *Maude-NPA* es una herramienta que funciona con una búsqueda hacia atrás; es decir, que partiendo de un estado final dado por nosotros; el cual en nuestro caso va a ser que haya un intruso que haya conseguido sacar información de un usuario, encontremos un estado inicial de esa situación. En caso de que encontremos un caso inicial podremos decir que realmente ese escenario puede suceder y que evidentemente hay una vulnerabilidad. Por otro lado, en caso de que no encontremos un estado inicial podemos ratificar que ese ataque no va a ocurrir en ninguna de las ejecuciones.

Antes de comenzar con el desarrollo vamos a ver en que consiste la especificación de un protocolo en la herramienta. [8]

4.2 Especificación en Maude-NPA

La especificación de un protocolo en la herramienta se divide en tres bloques principales: la sintaxis del protocolo, las propiedades algebraicas y la definición del comportamiento del protocolo.

4.2.1 Sintaxis del protocolo

La sintaxis del protocolo se define en el módulo PROTOCOL-EXAMPLE-SYMBOLS. *Maude-NPA* se alimenta de tres tipos de datos, de los que posteriormente se desarrollarán estructuras de datos mayores. Estos son:

- *Msg*: es la estructura básica de datos de la herramienta, cualquier tipo de datos que necesitemos definir deberá ser un subtipo de *Msg* y jamás habrá ninguna estructura que sea supertipo de *Msg*.
- *Public*: estructura creada para definir términos de uso y conocimiento público. No puede tener supertipos.
- *Fresh*: estructura de datos que identifica que el valor definido debe ser único en cada ejecución. No puede tener subtipos.

En este bloque también debemos definir los operadores del protocolo, se pueden definir como infijos; situados entre dos términos, o prefijos; situados antes del término.

```
--- Encoding operators for public/private encryption
op pk : Key Msg -> Msg [frozen] .
op sk : Key Msg -> Msg [frozen] .
```

Figura 3. Ejemplo de operadores en Maude-NPA

En la figura 3, podemos observar un ejemplo de la definición de los operadores, en este caso las funciones de encriptación y desencriptación, ya que son propiedades comunes.

4.2.2 Propiedades algebraicas

En este bloque debemos de definir las propiedades específicas del protocolo que estemos analizando. Estas propiedades se definen bajo las reglas de unificación y de las reglas de reescritura.

```
eq exp(exp(W:Gen,Y:NeNonceSet),Z:NeNonceSet)
   = exp(W:Gen, Y:NeNonceSet * Z:NeNonceSet) [variant] .
  eq e(K:Key,d(K:Key,M:Msg)) = M:Msg [variant] .
  eq d(K:Key,e(K:Key,M:Msg)) = M:Msg [variant] .
```

Figura 4. Ejemplo de la propiedad algebraica

de la exponenciación en Maude-NPA

Como ejemplo vamos a utilizar la propiedad de la exponenciación del algoritmo de *Diffie-Helman*, donde usando las reglas de reescritura y las reglas de concatenación.

4.2.3 Comportamiento del protocolo

Tanto el protocolo como las habilidades del intruso se definen en le modulo PROTOCOL-SPECIFICATION, y se definen mediante *Strands* o procesos.

Los *Strands* son una secuencia de envío y recepción de mensajes, identificado como positivo y negativo, y que definen la ejecución del protocolo o las habilidades del intruso. Se expresan de la siguiente forma:

```
:: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ]
```

Figura 5. Ejemplo de Stand

Aquí podemos ver la recepción de dos mensajes, *X e Y* y el envió de la concatenación de ambos mensajes mediante el símbolo ; que llama a la función concatenación que deberíamos tener descrita.

En este mismo bloque especificaremos también las posibles vulnerabilidades que pudiera tener el intruso, conocido como *attack patterns*. En el que definiremos el estado final que buscamos, es decir, un estado en el que el intruso obtenga la información que desea.

```
eq ATTACK-STATE(0) =
  :: r ::
  [ nil,
     -(pk(b,a ; N)),
     +(pk(a, N ; n(b,r))),
     -(pk(b,n(b,r))) |
     nil ]
  ||
  n(b,r) inI
  ||
  nil
  ||
  nil
  ||
  nil
[nonexec] .
```

Figura 6. Ejemplo de Attack-State en Maude-NPA

En este caso vemos los *Strands* que identifican los conocimientos que debe de poseer el intruso en la última ejecución del protocolo. Pero esto no es todo lo que tenemos que definir cuando hablamos del intruso, ya que aquí solamente vemos lo que esperamos que conozca como última instancia.

Para poder permitir que *Maude-NPA* pueda hacer una búsqueda hacia atrás, primero debe de conocer cuáles son las capacidades del intruso, que tienen que ser definidas en un apartado conocido como *Dolev-Yao Strands* de la Figura 7.

```
eq STRANDS-DOLEVYAO
 = :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] &
   :: nil :: [ nil | -(X ; Y), +(X), nil ] &
   :: nil :: [ nil | -(X ; Y), +(Y), nil ] &
   :: nil :: [ nil | -(X), +(sk(i,X)), nil ] &
   :: nil :: [ nil | -(X), +(pk(Ke,X)), nil ]
[nonexec] .
```

Figura 7. Ejemplo de los Strands Dolevyao del intruso.

De esta forma, tras la definición de todas las variables y propiedades que componen el protocolo a analizar ya podríamos proceder con el análisis.

5. Desarrollo

En este apartado se detallará el procedimiento para definir el protocolo *DP-3T* sobre la herramienta de *Maude-NPA*, además de las trazas de ataque que se han propuesto y poder concluir si es seguro ante esas situaciones.

5.1 Sintaxis de DP-3T

Para poder definir el protocolo de la forma correcta, necesitamos crear nuevos tipos de datos y operadores.

```
--- Sort Information
sorts Name Timestamp EphimeralKey PrivateKey .
subsorts Timestamp Name EphimeralKey PrivateKey  < Msg .
subsort Name < Public . --- This is quite relevant and necessary

--- Nonce operator
op t : Name Fresh -> Timestamp [frozen] .

--- Intruder
ops u u1 u2 u3 s s1 s2 s3 i : -> Name .

--- Ephimeral
op eph : Name -> EphimeralKey [frozen] .

--- Keys
op sk : Name -> PrivateKey [frozen] .

--- Concatenation
op _;_ : Msg Msg -> Msg [frozen gather (e E)] .
```

Figura 8. Definición de la sintaxis del DP-3T

Como vemos en la Figura 8, en este caso concreto ha habido que crear los shorts *Timestamp*; usado para la variable de tiempo que comparten los usuarios, *EphimeralKey*; para definir todos los identificadores efímeros y, por último, *PrivateKey*; para definir todas las claves secretas de los usuarios. Todos estos *sorts* que hemos declarado nuevos deben de ser declarados *subsorts* de la clase principal *Msg*.

A continuación, debemos declarar los operadores que usaremos posteriormente en los *Strands*.

Lo primero es generar un operador para cada agente que va a aparecer en la comunicación más adelante; vamos a definir como nombres a u, u1, u2 y u3 como todas las variantes de usuarios que vamos a necesitar; s1, s2 y s3 como todas las variantes del servidor y por último la i para identificar al intruso. Es necesario generar un operador para todos los identificadores efímeros, definido como *Ephimeral*, con el operador *eph*.

Para la clave secreta necesitamos definir el operador *Keys* como *sk***.**

También necesitamos un operador de concatenación, mediante el operador ; el cual nos permite dado un mensaje A y otro B, con el uso del operador A ; B conseguimos un mensaje con ambos.

5.2 Propiedades algebraicas de DP-3T

En este apartado, vamos a definir las propiedades algebraicas necesarias para el funcionamiento del protocolo DP-3T.

En este protocolo no tenemos ninguna propiedad algebraica a definir. Esto se debe a que los mensajes no van encriptados. Todos los mensajes se envían sin ningún tipo de encriptación por el canal público.

Realmente esto se debe a que la información que se envía es aleatoria, ya que viene derivada de la Clave Secreta del usuario, y solamente aquellos que tengan todos los datos pueden regenerar el *Sk* original.

Para aumentar la seguridad, en el momento en el que un usuario notifica que es positivo, elimina la *Sk* que tenía y genera una nueva, de forma que nadie sea capaz de rastrear al usuario.

5.3 Comportamiento del protocolo DP-3T

Este bloque es el más denso de todo el protocolo, aquí es donde vamos a definir tanto el funcionamiento del protocolo, como las habilidades del intruso y por último las trazas de ataque.

Vamos a comenzar por definir el funcionamiento del protocolo dentro del PROTOCOL-SPECIFICATION.

5.3.1 Especificación protocolo DP-3T

Para comenzar con la especificación del protocolo, debemos de establecer las variables que vamos a utilizar en la definición de los *Strands* del protocolo.

```
vars A B C D : Name .
vars r r' r1 r2 r3 : Fresh .
vars EphID EphIDA EphiIDB : EphimeralKey .
vars Sk SkA SkB : PrivateKey .
vars M M1 M2 : Msg .
var T : Timestamp .
```

Figura 9. Constantes definidas para DP-3T

Definimos A, B, C y D como variables de tipo *Name* para identificar a todos los usuarios que intervienen en la comunicación.

En el caso del tipo *Fresh*, vamos a definir las variables r, r', r1, r2 y r3 que usaremos más adelante para nombrar claves y tokens de tiempo que necesitemos.

A nivel de mensajes, vamos a definir tanto M1, M2 y M para mensajes cualquiera que puedan intercambiar los interlocutores.

Como variables de *EphimeralID*, definimos *EphID*, *EphIDA* y *EphIDB* para identificar los mensajes que contienen el *EphID* de cada uno de los usuarios que intervienen en la comunicación.

En el apartado de secretos, vamos a definir la variable Sk, para la clave secreta por defecto, *SkA* para la clave secreta de A y *SkB* para la *clave* secreta de B.

Por último, definimos T como el Timestamp que usaremos para el envío del mensaje que contiene la fecha vinculada al *EphID*.

5.3.2 Strands del protocolo DP-3T

En este apartado, vamos a especificar los *Strands* del protocolo; es decir, el comportamiento de los participantes honestos.

Los *Strands* se definen como roles que pueden tener cada uno de los usuarios en el proceso de la comunicación. Dentro de cada rol, debemos de definir las funciones que pueden realizar cada uno de los usuarios para el correcto funcionamiento del protocolo.

```
eq STRANDS-PROTOCOL =
  :: r ::   --- Role User Infected
  [nil | +(User1 ; User2 ; eph(User1)),
         -(User2 ; User1 ; EphUser2),
         +(User1 ; Serv ; sk(User1) ; t(User1,r)), nil ]
  &
  :: nil :: --- Role User Notified
  [nil | -(User1 ; User2 ; EphUser1),
         +(User2 ; User1 ; eph(User2)),
         -(Serv ; SkUser1 ; T), nil ]
  &
  :: nil :: --- Role Server
  [nil | -(User1 ; Serv ; SkUser1 ; T),
         +(Serv ; SkUser1 ; T), nil ]
[nonexec] .
```

Figura 10. Strands de la definición del protocolo DP-3T

Como vemos en la Figura 10, vamos a definir tres *Strands*, es decir, para el correcto funcionamiento del protocolo, necesitamos que intervengan tres usuarios con roles diferentes, también conocido como protocolo a tres bandas.

Entre los roles que vamos a definir, tenemos al usuario infectado, que es el usuario que tras intercambiar su *Ephimeral* ID va a reportarse al Servidor, para notificar que es un caso positivo de COVID-19. También tenemos el rol de usuario notificado; cuya función consiste en ser contacto del usuario que tiene el rol de infectado, de esta forma, tras el intercambio de información debe ser notificado por el servidor que ha sido contacto.

Por último, hemos definido el rol del servidor, cuya función es de transmisor de la información, los datos proporcionados por el usuario infectado los pasa al usuario que ha sido contacto.

En cierto momento de la ejecución del código, necesitamos que el servidor haga un Broadcast o Difusión, reenviando los datos recibidos por el usuario infectado a todo el resto de los usuarios. Para poder expresar este envío, tenemos que configurarlo de una forma diferente. De forma habitual, solemos definir un mensaje con la siguiente estructura.

Para mensajes enviados:

+(UsuarioEnvia ; UsuarioRecibe ; Mensaje)

Para mensajes recibidos:

-(UsuarioEnvia ; UsuarioRecibe ; Mensaje)

Pero cuando tenemos que definir un envío en Broadcast, el usuario que recibe debe de quedar en blanco, simbolizando que se envía a cualquiera, quedando de la siguiente forma.

Para mensajes enviados:

+(UsuarioEnvia ; Mensaje)

Para mensajes recibidos:

-(UsuarioEnvia ; Mensaje)

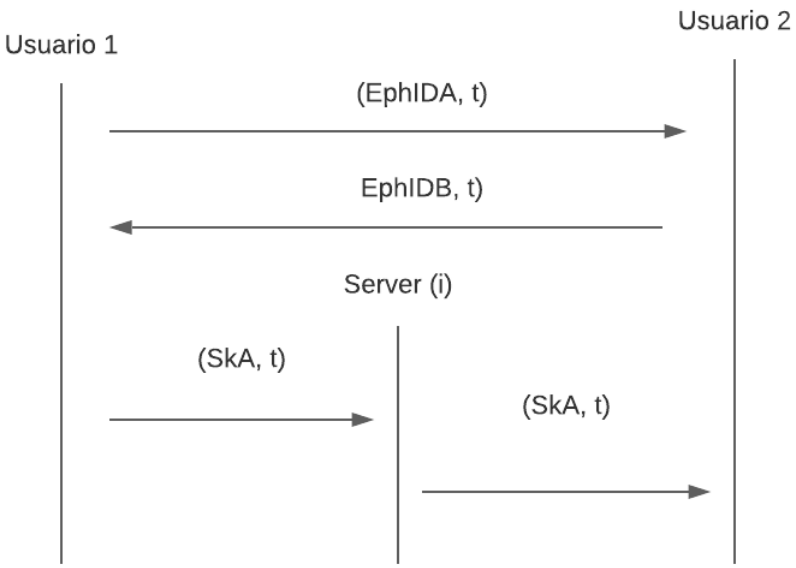

Figura 11. Ejemplo de comunicación del protocolo.

Los usuarios A y B han estado en contacto, por lo que de forma automática intercambian su *EphID* y el *Timestamp* t, identificando el momento en el que se hizo la transferencia de ese dato. Recordemos, que el *Ephimeral* ID es un identificador aleatorio generado a partir de la clave secreta de cada uno de los usuarios.

En el momento en que el usuario A comunica que ha dado positivo en una prueba de COVID-19, lo comunica por la aplicación. Esto se traduce en el usuario A enviando su Clave Secreta Sk y el *Timestamp* t al Servidor; indicándole que es un usuario positivo. En este momento, el servidor realiza un Broadcast de el mismo mensaje que le ha enviado el usuario A, es decir, reenvía la Sk de A y el *Timestamp* de A.

Este mensaje es recibido por todos los usuarios que utilizan la aplicación. En el caso que estamos tratando en la Figura 11, el mensaje llega al usuario B, que ha sido contacto del usuario A reportado como positivo.

A partir de este momento, el usuario B, mediante la *Sk* de A y su *Timestamp*, es capaz de reproducir si en algún momento ha recibido un *EphID* del usuario A, en cuyo caso significaría que ha sido contacto. Todo este proceso lo omitimos de la ejecución el protocolo en la herramienta, ya que al ser una computación interna del usuario y ya haber finalizado la transferencia de datos, no es sensible para el estudio.

A continuación, vamos a proceder con la especificación de las habilidades del intruso a través de los *Dolev-Yao Strands*. Este paso es necesario para poder definir luego las trazas de ataque y permitir a la herramienta realizar la ejecución hacia atrás.

```
eq STRANDS-DOLEVYAO =
   :: nil :: [ nil | -(M1 ; M2), +(M1), nil ] &
   :: nil :: [ nil | -(M1 ; M2), +(M2), nil ] &
   :: nil :: [ nil | -(M1), -(M2), +(M1 ; M2), nil ] &
   :: nil :: [ nil | +(eph(i)), nil ] &
   :: nil :: [ nil | +(sk(i)), nil ] &
   :: r ::   [ nil | +(t(i,r)), nil ] &
   :: nil :: [ nil | +(A), nil ]
[nonexec] .
```

Figura 12. Strands Dolevyao del DP-3T

Como podemos apreciar en la Figura 12, vamos a definir las habilidades que puede realizar.

- La primera es la habilidad de enviar solamente la primera parte de un mensaje recibido que concatena dos mensajes.
- La segunda habilidad permite enviar solamente la segunda parte de un mensaje recibido que concatena dos mensajes.
- La tercera habilidad es más compleja, permite al intruso enviar dos mensajes concatenados en un único mensaje a partir de dos mensajes recibidos por separado.
- La cuarta habilidad consiste en poder enviar un mensaje con el *Ephimeral ID* propio del intruso como *eph*(i).
- La quinta habilidad permite enviar un mensaje con la Clave Secreta *Sk* del intruso como *sk*(i).
- Como sexta habilidad permitimos al intruso poder enviar un *Timestamp* t propio al resto de usuarios.
- La séptima y última habilidad que concedemos al intruso es la posibilidad de suplantar la identidad de otros usuarios durante la comunicación.

Por último, tenemos que definir los posibles ataques que queremos analizar, conocido como *Attack-states*. En este caso vamos a realizar una primera ejecución que va a servir para comprobar que la comunicación es funcional con la definición del protocolo que hemos realizado, los siguientes van a ser las posibles vulneraciones que vamos a estudiar como finalidad de este trabajo.

5.3.3 Posibles vulnerabilidades que puede explorar un intruso

Vamos a comenzar con la definición de las trazas de ataque o *attack-states*. Los *attack-states* son los estados que definimos como el estado final del protocolo que pretendemos alcanzar. Partiendo de este estado final, en conjunto con la definición del protocolo y las habilidades definidas para el intruso, la herramienta comenzará a realizar una búsqueda hacia atrás tratando de encontrar si existe un estado inicial que desate toda esta traza.

El resultado de todos estos ataques se describe en el capítulo seis.

5.3.3.1 Attack-state(0)

Comenzamos con la definición del ATTACK-STATE(0), que como hemos comentado anteriormente va a ser una ejecución regular del protocolo para comprobar que la comunicación funciona con la definición que hemos realizado del mismo.

```
eq ATTACK-STATE(0) --- Regular execution
 = :: r ::   --- Role User Infected
   [nil,  +(u1 ; u2 ; eph(u1)),
          -(u2 ; u1 ; eph(u2)),
          +(u1 ; s ; sk(u1) ; t(u1,r)) | nil ]
   &
   :: nil :: --- Role User Notified
   [nil,  -(u1 ; u2 ; eph(u1)),
          +(u2 ; u1 ; eph(u2)),
          -(s ; sk(u1) ; t(u1,r)) | nil ]
   &
   :: nil :: --- Role Server
   [nil,  -(u1 ; s ; sk(u1) ; t(u1,r)),
          +(s ; sk(u1) ; t(u1,r)) | nil ]
   || empty
   || nil
   || nil
   || nil
[nonexec] .
```

Figura 13. Attack-State de la ejecución regular

Con este primer *Attack-state*, vamos a comprobar que la ejecución principal del código es correcta. Mediante la creación de un estado final deseado que sea el mismo que el estado que hemos configurado como estado final correcto (es decir, al que tendría que llegar en cualquier ejecución según el protocolo) vamos a comprobar que es correcta la definición que hemos hecho del mismo.

En este caso ha sido sencillo configurar este estado, ya que está basado en el mismo *STRAND-PROTOCOL*, principalmente se trata de sustituir las variables por expresiones y añadir los operadores correspondientes.

En este caso concreto, los *STRANDS-DOLEVYAO*, que indican las habilidades del intruso no se utilizan, ya que en la ejecución regular no existe el intruso, simplemente la comunicación entre los usuarios y el servidor.

Para indicar el final del *Attack State*, el código cierra con tres *nil*, que implican que no se espera mensaje y por último con *[noexec]*, una información complementaria requerida por *Maude-NPA*.

5.3.3.2 Attack-state (1)

Para la definición del ATTACK-STATE (1) vamos a emular un ataque de *Backend Impersonation* o Suplantación del servidor.

En esta situación tenemos que configurar un estado final en la que el intercambio de los *Ephimeral* ID de cada usuario se realiza de forma normal, pero en el momento en el que el usuario A debe reportar al Servidor que ha sido positivo, la comunicación que pretende establecer contra el Servidor, lo realiza contra el intruso. Con esta suplantación, el intruso pretende averiguar el Sk y el *timestamp* que posee el usuario

El intruso debe realizar las siguientes acciones:

- Elegir un *Sk* y transformarlo en un *EphID* (este paso lo omitimos, ya que el servidor de forma recurrente envía los *Sk* de los infectados a todos los usuarios).
- Acercarse a la víctima (esto lo marcamos dentro del contexto del ataque)
- Enviar el *EphID* generado a partir del *Sk*.
- Suplantar al servidor para recibir el *Sk* del usuario cuando este vaya a reportar que es un caso positivo, ya que este es el momento en el que envía su *Sk* en lugar de su *EphID*.

```
eq ATTACK-STATE(1) --- Backend impersonation
 = :: r ::    --- Role User Infected
   [nil,  +(u1 ; u2 ; eph(u1)),
          -(u2 ; u1 ; eph(u2)),
          +(u1 ; i ; sk(u1) ; t(u1,r)) | nil ]
   &
   :: nil :: --- Role User Notified
   [nil,  -(u1 ; u2 ; eph(u1)),
          +(u2 ; u1 ; eph(u2)),
          -(i ; sk(u1) ; t(u1,r)) | nil ]
   &
   :: nil :: --- Role Server
   [nil,  -(i ; s ; sk(u1) ; t(u1,r)),
          +(s ; sk(u1) ; t(u1,r)) | nil ]
   || empty
   || nil
   || nil
   || nil
[nonexec] .
```

Figura 14. Attack-State de Suplantación del Servidor

En la Figura 14, podemos apreciar la traza que hemos realizado. En este ataque, debemos de mantener los tres roles que definimos en la definición. El rol de usuario infectado, el rol de usuario notificado y el rol servidor.

Para el Usuario 1, con rol de infectado, definimos un intercambio de *EphID* entre el Usuario 1 y el Usuario 2. Después de este intercambio, el Usuario 1 debería reportar al servidor que es positivo en COVID-19, por lo que envía su *Sk* y su *Timestamp* al servidor, pero en esta traza, analizamos si es posible que el intruso sea el que reciba esta información suplantando al servidor.

Para el Usuario 2, con rol de notificado, definimos el mismo intercambio de *EphID* que hemos realizado con el Usuario 1, y a continuación debería recibir el *Sk* y el *Timestamp* de A a través del servidor, pero en este caso debería de hacerlo el intruso, ya que lo está suplantando.

Por último, para el rol del servidor, hemos configurado el intercambio de información que realiza, es decir, recibe el *Sk* de A y el t de A y lo envía mediante un Broadcast al resto de usuarios. Pero en este caso, el intruso es quien está haciendo la función del servidor, así que es él quien recibe y envía estos datos.

Teniendo ya el conocimiento de lo que hace la traza, tenemos que analizar con la herramienta buscando al menos un estado inicial cuyo estado final tenga al intruso suplantando al servidor, que es el caso que hemos definido aquí.

5.3.3.3 Attack-state (2)

Para el segundo ATTACK-STATE vamos a emular un ataque de *False Report* o Falso Reporte.

Este ataque se basa en que el intruso reporte falsamente que es positivo al servidor para que todos aquellos contactos que haya tenido sean reportados. Por supuesto, para que este ataque tenga sentido, primero el intruso debe de intercambiar su *EphID* con la víctima.

En este ataque, el intruso tiene que asumir el rol de usuario infectado, a diferencia del ataque anterior, para ello, siguiendo la sintaxis del protocolo especificado, el intruso hará la función del Usuario 1, por lo que es importante definir en este ataque que u1 = i.

En resumen, el intruso debe:

- Escoger un *Sk* propio (no incluido la traza del ataque ya que no afecta a la comunicación).
- Derivar un *EphID* del *Sk* elegido (tampoco se incluye en la traza del ataque).
- Acercarse físicamente a la víctima (forma parte del contexto del ataque).
- Realizar el intercambio de *EphID* con la víctima.
- Reportar al servidor un caso positivo falso.

```
eq ATTACK-STATE(2) --- False report
 = :: r ::   --- Role User Infected
   [nil,  +(i ; u2 ; eph(i)),
          -(u2 ; i ; eph(u2)),
          +(i ; s ; sk(i) ; t(i,r)) | nil ]
   &
   :: nil :: --- Role User Notified
   [nil,  -(i ; u2 ; eph(i)),
          +(u2 ; i ; eph(u2)),
          -(s ; sk(i) ; t(i,r)) | nil ]
   &
   :: nil :: --- Role Server
   [nil,  -(i ; s ; sk(i) ; t(i,r)),
          +(s ; sk(i) ; t(i,r)) | nil ]
   || empty
   || nil
   || nil
   || nil
[nonexec] .
```

Figura 15. Attack-State para Reporte Falso.

En la Figura 16, podemos ver la traza que hemos realizado para este ataque en concreto.

El intruso, en el rol de usuario infectado, realiza el intercambio de *EphID* con el Usuario 2. Una vez ha realizado el intercambio y la victima ya tiene el *EphID* del intruso, procede a notificar al servidor un falso positivo, enviando al Server su propio *Sk* y el *Timestamp* con la fecha de creación.

En la situación del Usuario 2, en rol de usuario notificado, realiza el intercambio de su *EphID* con el intruso. Tras esto, el Usuario 2 recibe el Broadcast realizado por el servidor donde recibe el *Sk* del intruso y su *Timestamp*. Tras esto, el Usuario 2 descubrirá que es contacto una vez derive

el *EphID* del intruso con los datos recibidos (esto no se incluye en la ejecución ya que no influye en la comunicación).

Por último, el Servidor realiza el rol de servidor, recibiendo los datos del positivo (falso positivo en este caso) y haciendo un broadcast de esta información a todos los usuarios.

Una vez introduzcamos esta traza de ataque en la herramienta, podremos comprobar si existe algún estado inicial que justifique la existencia del ataque final que hemos definido.

5.3.3.4 Attack-state (3)

Para el tercer *Attack-state*, vamos a evaluar la vulnerabilidad *Reply of Released Cases* o Reenvío de Casos Liberados.

Esta vulnerabilidad consiste en que el intruso obtenga el *Sk* y el *Timestamp* de uno de los usuarios que han sido positivos. El intruso debe de poder obtener estos datos del Broadcast realizado por el servidor. Con esta información en su poder, el intruso puede derivar la información del usuario positivo en un *EphID* para ir enviándolo a los usuarios con los que contacte.

El procedimiento del ataque se define de la siguiente manera:

- El intruso recoge los datos de un usuario positivo.
- El intruso deriva un *EphID* con los datos recibidos (esta funcionalidad no la implementamos ya que no influye en el proceso de comunicación).
- El intruso envía el *EphID* a otros usuarios (este proceso tampoco lo analizamos ya que esta funcionalidad sabemos que si lo puede lograr).

```
eq ATTACK-STATE(3) --- Reply of infected case
 = :: r ::   --- Role User Infected
   [nil,  +(u1 ; u2 ; eph(u1)),
          -(u2 ; u1 ; eph(u2)),
          +(u1 ; s ; sk(u1) ; t(u1,r)) | nil ]
   &
   :: nil :: --- Role User Notified
   [nil,  -(u1 ; u2 ; eph(u1)),
          +(u2 ; u1 ; eph(u2)),
          -(s ; sk(u1) ; t(u1,r)) | nil ]
   &
   :: nil :: --- Role Server
   [nil,  -(u1 ; s ; sk(u1) ; t(u1,r)),
          +(s ; sk(u1) ; t(u1,r)) | nil ]
   || (sk(u1) ; t(u1,r)) inI
   || nil
   || nil
   || nil
[nonexec] .
```

Figura 16. Attack-State para Reenvío de Casos.

Como vemos en la Figura 16, nos cuestionamos si es posible que, tras la ejecución del protocolo, el intruso haya podido obtener los datos que necesita para derivar un *EphID* con el que infectar a otros usuarios, es decir, el *Sk* del infectado y su *timestamp*.

En esta traza concreta, mantenemos la estructura de tres *Strands* que estamos manteniendo en las últimas ejecuciones. El Usuario 1 tiene el rol de infectado, donde intercambia el *EphID* con el Usuario 2 y después notifica al servidor que es positivo.

El Usuario 2 tiene el rol de notificado, donde hace el intercambio de *EphID* con el Usuario 1 y después recibe los datos del usuario que ha dado positivo.

El Servidor tiene su propio rol, recibiendo los datos del usuario positivo y haciendo Broadcast de los datos a todos los usuarios.

En esta traza, ya que no hemos tenido la interacción en ninguna de los *Strands*, indicamos que cuando termine la ejecución del estado final el intruso conoce el *Sk* y el *timestamp* del usuario positivo. Esto queda definido con la función:

(sk(u1) ; t(u1,r)) inI

Función que indica que el intruso debe de conocer esos datos al finalizar la ejecución.

Cuando lo analicemos, comprobaremos si existe un caso inicial en el que el intruso conozca los datos del usuario positivo cuando acabe.

5.3.3.5 Attack-state (4)

En el cuarto *Attack-state*, vamos a analizar el *Reply Attack* o Ataque de Reproducción, que consiste en la posibilidad del intruso para recolectar diferentes *EphiD* de diferentes usuarios, anticipando que alguno de ellos va a ser positivo. Al final se reduce a una cuestión estadística, cuantos más contactos almacenen, mayor probabilidad de ser contacto.

El procedimiento que define el protocolo es el siguiente:

- El intruso recoge varios *EphID* de otros usuarios.
- El intruso se acerca a la víctima (se establece como contexto, no se representa).
- El intruso envía los *EphID* recolectados a otros usuarios (no lo vamos a representar ya que solo estamos interesados en la capacidad del usuario de recolectar *EphID*).

```
eq ATTACK-STATE(4) --- Reply attack
 = :: r ::   --- Role User Infected
   [nil,  +(u1 ; i ; eph(u1)),
          -(i ; u1 ; eph(i)) |
          +(u1 ; i ; s ; sk(u1) ; t(u1,r)), nil ]
   || (eph(u1) ; eph(i)) inI
   || nil
   || nil
   || nil
[nonexec] .
```

Figura 17. Definición del Attack-State (4)

En este ataque, solamente necesitamos un único *Strand*, que hace referencia al rol de usuario notificado.

El Usuario 1 realiza un intercambio de *EphID* con el intruso mediante el envío y recepción de los mensajes especificados. Esta es realmente la parte que nos interesa del *Strand* del primer rol, por lo que lo conveniente sería que la herramienta no tuviera que simular el siguiente envío. Para que no siga ejecutando partes del código que no necesitamos lo indicamos con el símbolo | el cual indica que todo el código que quede en la parte derecha no será ejecutado.

Es necesario indicar que al final de la ejecución, esperamos que el intruso posea en su histórico el *eph*(u1) y *eph*(i). Después del análisis, veremos si realmente es posible encontrar un estado inicial a partir del estado final propuesto.

5.3.3.6 Attack-state (5)

Como último ataque, en el estado 5 vamos a analizar un Relay Attack o Ataque por Retransmisión consiste en usar las habilidades del intruso para recibir un EphID de un usuario que sabemos que es positivo y enviar este mismo EphID a la víctima.

Este ataque se define de la siguiente manera:

- El intruso se aproxima a un usuario que sabe que es positivo.
- Ambos usuarios intercambian sus *EphID*.
- El intruso se acerca a la víctima.
- El intruso le envía a la víctima el *EphID* del usuario que es positivo.

```
eq ATTACK-STATE(5) --- Relay attack
 = :: r ::   --- Role User Infected
   [nil,  +(u1 ; i ; eph(u1)) |
          -(i ; u1 ; eph(i)),
          +(u1 ; i ; s ; sk(u1) ; t(u1,r)), nil ]
   &
   :: nil :: --- Role User Notified
   [nil,  -(i ; u2 ; eph(u1)),
          +(u2 ; u1 ; eph(u2)) |
          -(s ; u2 ; sk(u1) ; t(u1,r)), nil ]
   || eph(u2) inI
   || nil
   || nil
   || nil
[nonexec] .
```

Figura 18. Definición del Attack-State (5)

En este último ataque, también vamos a definir dos Strands, uno con rol de infectado y otro con rol de receptor.

Para el rol de infectado vamos a utilizar la variable u1 para identificar al usuario positivo. Del primer Strand, necesitamos el primer mensaje, emitido por el usuario positivo hacia el intruso, ya que han estado en contacto. El resto de la ejecución ya no es importante para el análisis, ya que solamente necesitamos que el intruso se haga con el EphID del positivo. Como en el resto de las ejecuciones, lo indicamos con el símbolo "|".

El otro Strand cumple el rol del receptor. El Usuario 2 y el intruso entran en contacto y comienza el intercambio de información; recibe un mensaje del intruso, que contiene el EphID del usuario positivo y el Usuario 1 envía su EphID. Como en el S trand anterior, ya no necesitamos procesar el resto del código, así que lo indicamos con la finalidad de optimizar el análisis del ataque.

Una vez ha terminado la ejecución del código, el Usuario 2 tiene que haber recibido el EphID del usuario positivo y el intruso ha sido el intermediario entre los dos usuarios. Para cerciorarnos que esto se cumple, indicamos que al acabar la comunicación el intruso tiene que conocer el EphID de Usuario 2, lo que indica que para ello han tenido que intercambiar datos entre intruso y Usuario 2, obligando a que la comunicación entre los dos suceda.

En el análisis comprobaremos si encontramos un estado inicial a partir del estado final que hemos definido aquí.

6. Resultados

Una vez ya hemos desarrollado todos los ataques que vamos a probar, debemos realizar la simulación en la herramienta, para ver los resultados y poder dar conclusiones sobre la seguridad del protocolo.

Vamos a revisar y a analizar primero la ejecución regular, que nos indicará si realmente existe comunicación con el protocolo que hemos definido

A continuación, vamos a analizar los ataques que hemos definido, comprobando si se cumple que con ese ataque se puede vulnerar el protocolo o no. Esta herramienta también permite certificar que un protocolo es seguro siempre que se alcance el resultado (0,0) que indica que no queda ningún estado hacia el que retroceder y que no ha encontrado ninguna solución

Recordemos que *Maude-NPA* verifica los protocolos mediante la búsqueda hacia atrás. Esta búsqueda hacia atrás la realiza en un árbol de búsqueda, comenzando por la última hoja del árbol y va profundizando hacia la raíz.

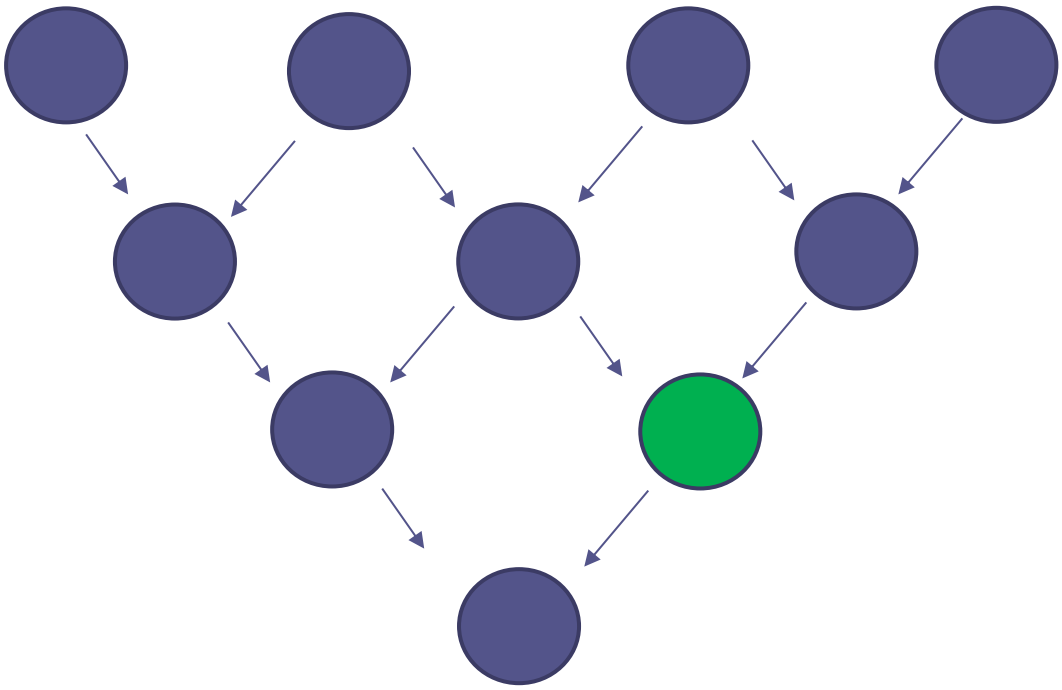

Figura 19. Ejemplo de Árbol de Búsqueda.

En la Figura 19 podemos observar una recreación del árbol de búsqueda. En la herramienta vamos a ejecutar el comando *summary*, el cual nos devuelve el número de estados encontrados en las hojas del árbol que aún puede alcanzar y cuantas de esas hojas son estados iniciales del ataque que estamos analizando.

En la función *summary*(x,y), la variable x se refiere al número del ATTACK-STATE y la variable y a la profundidad a la que queremos buscar.

Como caso de ejemplo:

Summary (0,1) → Primer nivel de profundidad. *States*>>4 *Solutions*>>0

Summary (0,2) → Segundo nivel de profundidad. *States*>>3 *Solutions*>>0

Summary (0,3) → Tercer nivel de profundidad. *States*>>2 *Solutions*>>1

En la tercera profundidad ya hemos encontrado un estado inicial que cumple ese estado final.

Ahora que ya conocemos como Maude-NPA devuelve los resultados, vamos a analizar los casos expuestos.

6.1 Resultado ejecución regular

Una vez con la traza de ataque definida, procedemos a ejecutar la consulta en la herramienta, la cual nos da los siguientes resultados:

```
red summary(0,0) . --- States>> 1  Solutions>> 0
red summary(0,1) . --- States>> 4  Solutions>> 0
red summary(0,2) . --- States>> 7  Solutions>> 0
red summary(0,3) . --- States>> 12 Solutions>> 0
red summary(0,4) . --- States>> 12 Solutions>> 1
red initials(0,4) .
```

Figura 20. Resultados de ejecución regular.

En la Figura 20, observamos los resultados. Tras ejecutar cinco búsquedas hacia atrás o llegar al nivel de profundidad cuatro del árbol, la herramienta ha sido capaz de encontrar una etapa final en la que se cumple. Es decir, hay un estado inicial para el estado final que habíamos configurado. Esto nos confirma que el protocolo está bien definido, ya que es capaz de realizar la transferencia de los datos sin problemas.

Además de esto, la herramienta nos indica que existen doce estados, por lo que si siguiéramos profundizado es posible que encontremos más de un estado inicial.

Vamos a ver la traza de la ejecución del protocolo desde el estado inicial encontrado por la herramienta hasta el estado final indicado.

```
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
-(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
+(s ; sk(u1) ; t(u1, #0:Fresh)),
-(s ; sk(u1) ; t(u1, #0:Fresh))
```

Figura 21. Traza de ejecución encontrada.

Como podemos ver en la Figura 21, la ejecución devuelta por la herramienta es igual a los Strands definidos en la especificación del protocolo, pero de una forma ordenada. Al final se trata del resultado esperado siempre y cuando hubiéramos definido bien el protocolo, ya que, en la ejecución regular, si la comunicación funciona, el estado inicial y el final coinciden.

Con este resultado no podemos probar la seguridad del protocolo de ninguna forma, para ello tenemos que analizar el resto de ataques que hemos definido, pero esta ejecución regular nos permite conocer una definición del protocolo que es funcional, que es fiable a la hora de verificar los demás ataques y que en este caso puede sentar una base sólida para futuras investigaciones sobre el protocolo DP-3T en *Maude*-NPA, ya que esta es la primera vez que se trata de verificar el protocolo DP-3T sobre esta.

6.2 Resultados primer ataque

En este momento comenzamos con la verificación de la seguridad del protocolo. Vamos a comprobar si el servidor es seguro ante un ataque de Suplantación del Servidor.

Este ataque estaba definido en el *ATTACK-STATE (1*), y tras ejecutarlo en la herramienta obtenemos los siguientes resultados:

```
red summary(1,0) . --- States>> 1  Solutions>> 0
red summary(1,1) . --- States>> 2  Solutions>> 0
red summary(1,2) . --- States>> 4  Solutions>> 0
red summary(1,3) . --- States>> 5  Solutions>> 0
red summary(1,4) . --- States>> 7  Solutions>> 0
red summary(1,5) . --- States>> 9  Solutions>> 0
red summary(1,6) . --- States>> 15 Solutions>> 0
red summary(1,7) . --- States>> 15 Solutions>> 1
red initials(1,7) .
```

Figura 22. Resultados del Attack-State 1.

Con los resultados obtenidos de la Figura 22, podemos determinar que el protocolo DP-3T es vulnerable ante un ataque de *Backend Impersonation*. Durante la búsqueda hacia atrás, en la séptima profundidad ya había encontrado un estado inicial, por consiguiente, el protocolo es vulnerable ante una suplantación por parte del servidor.

En la séptima profundidad, donde ya hemos encontrado un estado inicial, también tenemos 15 estados en ese nivel, por lo que si seguimos retrocediendo podríamos encontrar algún estado inicial más, pero con encontrar un estado inicial es suficiente para el análisis que estamos realizando.

Ahora vamos a ver la ejecución que ha encontrado la herramienta para este ataque.

```
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)),
-(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)),
+(i ; sk(u1) ; t(u1, #0:Fresh)),
-(i ; sk(u1) ; t(u1, #0:Fresh)),
+(sk(u1) ; t(u1, #0:Fresh)),
-(s),
-(sk(u1) ; t(u1, #0:Fresh)),
+(s ; sk(u1) ; t(u1, #0:Fresh)),
-(i),
-(s ; sk(u1) ; t(u1, #0:Fresh)),
+(i ; s ; sk(u1) ; t(u1, #0:Fresh)),
-(i ; s ; sk(u1) ; t(u1, #0:Fresh))
```

Figura 23. Ejecución obtenida de Attack-State (1).

En la Figura 23, podemos observar cómo tras el intercambio de información entre el Usuario 1 y el Usuario 2, el intruso intercepta el mensaje del Usuario 1 donde pretende indicar al servidor que es positivo.

Como el intruso debe de realizar las acciones del servidor, cuando recibe el Sk de Usuario 1, debe reenviarlo como broadcast a todos los usuarios, entre ellos el servidor, el cual, al recibir el mensaje también tiene que reenviarlo a todos, ya que su rol consiste en ello.

Es por esto por lo que vemos en la traza que tanto el intruso como el servidor hacen broadcast de la información.

Con la intención de hacerlo más visual, generamos un diagrama de interacción.

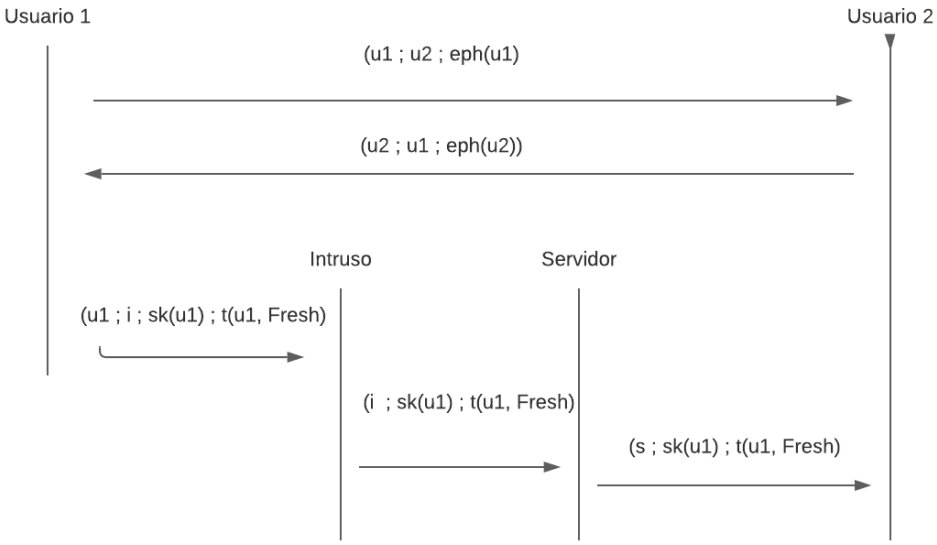

Figura 24. Attack-State 1 en un diagrama de interacción

6.3 Resultados segundo ataque

El segundo ataque se trataba del *False Report* o Reporte Falso, en este apartado vamos a verificar si el protocolo es seguro ante este ataque.

Este ataque estaba definido en el ATTACK-STATE (2), y tras ejecutarlo en la herramienta obtenemos los siguientes resultados:

```
red summary(2,0) . --- States>> 1  Solutions>> 0
red summary(2,1) . --- States>> 3  Solutions>> 0
red summary(2,2) . --- States>> 5  Solutions>> 0
red summary(2,3) . --- States>> 5  Solutions>> 1
red initials(2,3) .
```

Figura 25. Resultados del Attack-State 2.

Tras la ejecución de *Maude-NPA*, llegamos a la conclusión que en la tercera profundidad se ha encontrado un estado inicial que cumple el estado final del ataque de *False Report*. Podríamos continuar profundizando en el árbol de búsqueda para encontrar otro estado inicial, ya que en la profundidad actual se han encontrado 5 nodos, pero como solamente necesitamos encontrar un estado inicial para comprobar que no es seguro ante el ataque, no seguiremos buscando.

Vamos a visualizar la ejecución que ha alcanzado la solución.

```
+(i ; u2 ; eph(i)),
generatedByIntruder(i ; u2 ; eph(i)),
-(i ; u2 ; eph(i)),
+(u2 ; i ; eph(u2)),
-(u2 ; i ; eph(u2)),
+(i ; s ; sk(i) ; t(i, #0:Fresh)),
-(i ; s ; sk(i) ; t(i, #0:Fresh))
```

Figura 26. Traza de la ejecución de Attack-State (2).

Tal y como se presenta en la traza de la ejecución, vemos que el intruso ha podido generar un *EphID* propio, ya que se encontraba entre las habilidades que se le habían proporcionado. Tras un intercambio de sus *EphID* con el Usuario 2 y procede a reportarse con el servidor.

Tras esta ejecución, tras el reporte del intruso al servidor, el Usuario 2 acabará recibiendo el *Sk* del intruso y descubriendo que ha sido contacto. Esta parte la hemos omitido, ya que, para el análisis, una vez el intruso es capaz de reportar un falso positivo, el ataque ya es efectivo.

Con la intención de hacerlo más visual, generamos un diagrama de interacción.

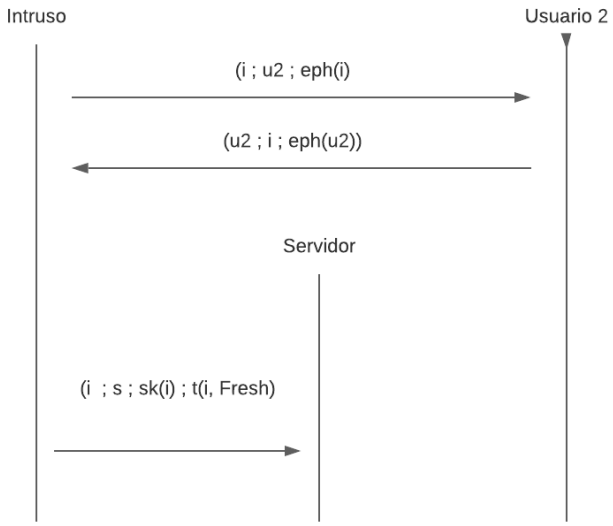

Figura 27. Attack-State (2) en un diagrama de interacción.

6.4 Resultados tercer ataque

El tercer ataque es el *Reply of Released Cases* o reenvío de Casos Relacionados, en este apartado de resultados, vamos a comprobar si el protocolo DP-3T es vulnerable a este ataque.

Este ataque estaba definido en el *ATTACK-STATE (2*), y tras ejecutarlo en la herramienta obtenemos los siguientes resultados:

```
red summary(3,0) . --- States>> 1  Solutions>> 0
red summary(3,1) . --- States>> 4  Solutions>> 0
red summary(3,2) . --- States>> 8  Solutions>> 0
red summary(3,3) . --- States>> 11 Solutions>> 0
red summary(3,4) . --- States>> 14  Solutions>> 0
red summary(3,5) . --- States>> 13  Solutions>> 1
red initials(3,5) .
```

Figura 28. Resultados del Attack-State 3.

Analizando los resultados obtenidos en la Figura 28, podemos confirmar que el protocolo tampoco es seguro ante un ataque de *Reply of Related Cases*. Hemos tenido que sumergirnos

hasta la quinta profundidad para poder encontrar una etapa inicial que conecte con la etapa final que hemos definido para este ataque. No vamos a seguir realizando búsquedas en el árbol ya que solamente necesitamos un estado inicial para poder confirmar que el protocolo no es seguro ante este ataque.

Vamos a revisar la ejecución que ha encontrado la herramienta.

```
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
-(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
+(s ; sk(u1) ; t(u1, #0:Fresh)),
-(s ; sk(u1) ; t(u1, #0:Fresh)),
+(sk(u1) ; t(u1, #0:Fresh)),
-(s ; sk(u1) ; t(u1, #0:Fresh))
```

Figura 29. Ejecución del Attack-State (3).

Esta ejecución es un poco más compleja de analizar. Para empezar, como siempre tenemos el intercambio de *EphIDs* entre Usuario 1 y Usuario 2. Después, el Usuario 1 se reporta al servidor indicando que es positivo. El servidor, cumpliendo con su rol, en el momento en el que recibe el *Sk* de un usuario que se reporta como positivo, reenvía estos datos a todos los usuarios.

Como en la definición de este ataque, no aparece el intruso en ninguno de los mensajes, tampoco lo hace en la traza, pero como habíamos definido que cuando acabara la ejecución el

intruso debía de conocer el *Sk* del Usuario 1 y se ha encontrado un estado que lo reafirma, podemos decir, que el intruso sí que posee el *Sk* del usuario positivo.

Realmente, aunque no esté marcado en la traza de la ejecución, el intruso ha debido de recibir los datos que buscaba en el momento en el que el servidor ha realizado el Broadcast de estos.

La traza continua con los reportes del servidor acerca de los usuarios infectados, pero como hemos indicado, esto no es sensible para nosotros a la hora de analizar esta traza.

Con la intención de hacerlo más visual, generamos un diagrama de interacción.

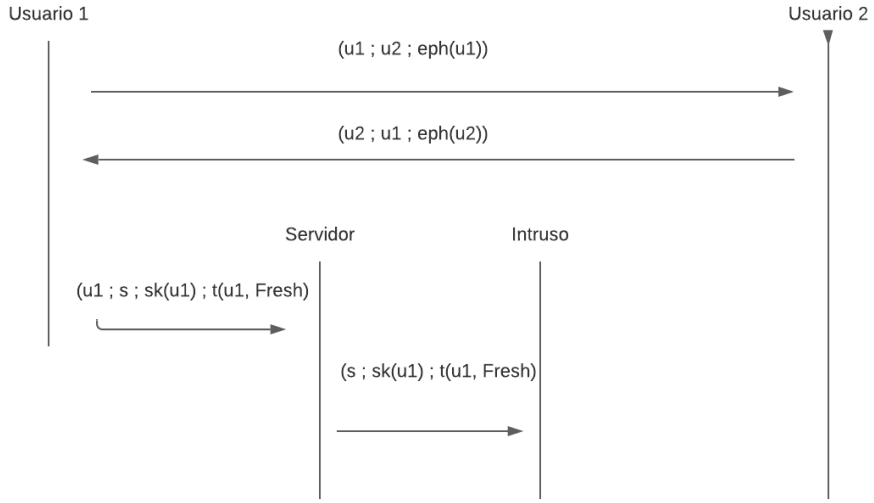

Figura 30. Attack-State (3) en un diagrama de interacción.

6.5 Resultados cuarto ataque

El cuarto ataque es el *Reply Attack* o Ataque de reenvío, en este apartado de resultados, vamos a comprobar si el protocolo DP-3T es vulnerable a este ataque.

Este ataque se vio analizado en el ATTACK-STATE (4), vamos a ver los resultados obtenidos en la herramienta.

```
red summary(4,0) . --- States>> 1  Solutions>> 0
red summary(4,1) . --- States>> 2  Solutions>> 0
red summary(4,2) . --- States>> 3  Solutions>> 0
red summary(4,3) . --- States>> 2  Solutions>> 0
red summary(4,4) . --- States>> 3  Solutions>> 1
red initials(4,4) .
```

Figura 31. Resultados del Attack-State 4.

Tras ver los resultados, podemos confirmar que el protocolo *DP-3T* tampoco es seguro ante un Ataque de Reenvío. Cuando alcanzamos la cuarta profundidad el protocolo encuentra un estado inicial que cumple la condición. A pesar de que en esta profundidad aún tenemos tres estados, una vez hemos encontrado al menos un estado inicial no nos interesa seguir buscando, ya que hemos demostrado que no es seguro.

Vamos a analizar la ejecución que ha encontrado la herramienta.

```
+(u1 ; i ; eph(u1)),
-(u1 ; i ; eph(u1)),
+(i ; eph(u1)),
-(i ; eph(u1)),
+(eph(u1)),
generatedByIntruder(eph(i)),
-(eph(u1)),
-(eph(i)),
+(eph(u1) ; eph(i)),
generatedByIntruder(i ; u1 ; eph(i)),
-(i ; u1 ; eph(i))
```

Figura 32. Ejecución del Attack-State(4).

En esta ejecución, comienza el intercambio del *EphID* de Usuario 1 con el intruso. El intruso es capaz de generar un *EphID* propio ya que le hemos concedido esa habilidad. Una vez tiene el identificador de Usuario 1, con el fin de concatenar los *EphID* que va recibiendo, realiza un broadcast de su propio identificador efímero, para poder recoger del canal de comunicación tanto el *EphID* de Usuario 1 como el del intruso, ya que la única forma que tiene de concatenar mensajes es recogiéndolos del canal.

Una vez ya ha conseguido concatenar estos identificadores los guarda y envía al Usuario 1 su identificador propio, cerrando así el envío de *EphID* de los usuarios.

Con la intención de hacerlo más visual, generamos un diagrama de interacción.

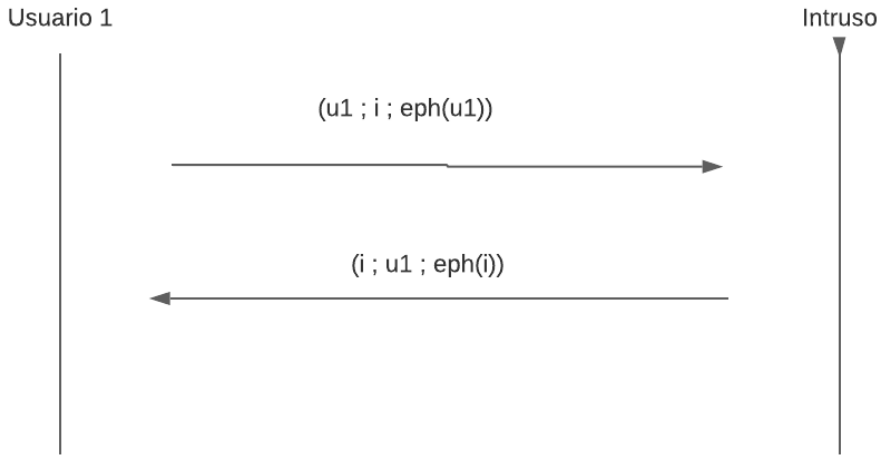

Figura 33. Diagrama de interacción del Attack-State(4).

Determino que es importante indicar que, aunque en el diagrama no se aprecie, el intruso ha concatenado el *EphID* de Usuario 1 junto al suyo para enviarlo al resto de usuarios con los que tenga contacto de ahora en adelante. En este caso no tenía sentido que el intruso le contestara con la concatenación de identificadores que tiene porque estaría contestando al Usuario 1 con su propio identificador.

6.6 Resultados quinto ataque

El quinto y último ataque es el *Relay Attack* o Ataque por Retransmisión. En este apartado vamos a analizar los resultados obtenidos al simular el ataque.

Este ataque se vio analizado en el *ATTACK-STATE (5*), vamos a revisar los resultados obtenidos.

```
red summary(5,0) . --- States>> 1  Solutions>> 0
red summary(5,1) . --- States>> 3  Solutions>> 0
red summary(5,2) . --- States>> 5  Solutions>> 0
red summary(5,3) . --- States>> 9  Solutions>> 0
red summary(5,4) . --- States>> 17 Solutions>> 0
red summary(5,5) . --- States>> 27 Solutions>> 0
red summary(5,6) . --- States>> 23 Solutions>> 0
red summary(5,7) . --- States>> 21 Solutions>> 2
red initials(5,7) .
```

Figura 34. Resultados del Attack-State(5).

Analizando el resultado obtenido, podemos comprobar que el protocolo DP-3T tampoco es seguro frente a un *Relay Attack*, ya que en la séptima profundidad hemos dos estados iniciales que cumplen el ataque.

A esta profundidad nos encontramos veintiún estados todavía, pero no vamos a profundizar más, ya que con encontrar al menos una etapa inicial tenemos información suficiente para el análisis.

Analicemos una de las ejecuciones del ataque.

```
generatedByIntruder(u2 ; u1 ; #1:EphimeralKey),
-(u2 ; u1 ; #1:EphimeralKey),
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; eph(u1)),
-(i),
-(u2 ; eph(u1)),
+(i ; u2 ; eph(u1)),
-(i ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; eph(u2)),
-(u1 ; eph(u2)),
+(eph(u2)),
+(u1 ; i ; eph(u1))
```

Figura 35. Ejecución del Attack-State(5).

En esta ejecución, el intruso comienza generando un mensaje suplantando a Usuario 2 para establecer la comunicación y el intercambio de identificadores entre Usuario 1 y el intruso.

Una vez el intercambio finaliza, el intruso intercepta el mensaje dirigido a Usuario 2 y suplanta al emisor, consiguiendo así el *EphID* de Usuario 1. El Usuario 2 intercambia su *EphID* con el Usuario 1 ya que han sido contacto y, por último, cambia su clave con el intruso, del que también ha sido contacto para acabar recibiendo el *EphID* de Usuario 1.

Tras esto, el identificador efímero del usuario infectado a llegado a la víctima usando al intruso como pasarela, ya que ha sido el que ha orquestado la comunicación entre los dos usuarios.

Con la intención de hacerlo más visual, generamos un diagrama de interacción.

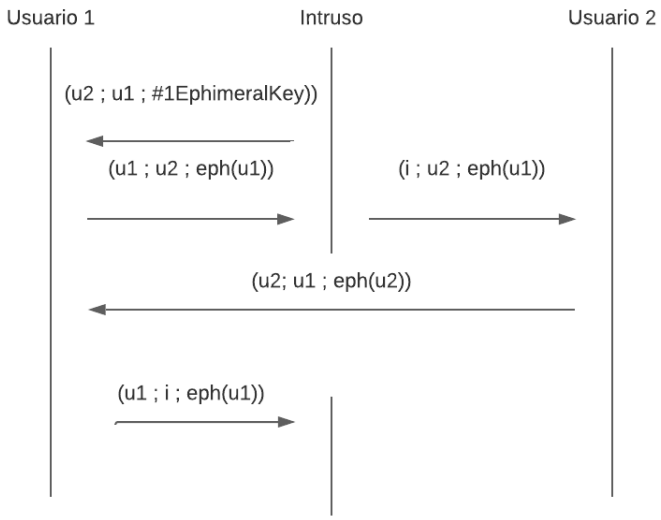

Figura 36. Ejecución del Attack-State(5) en diagrama de interacción.

7. Conclusión

En este apartado, se realizará una síntesis de todos los resultados analizados, tratando determinar el cumplimiento de los objetivos que se habían propuesto para el desarrollo del proyecto y se tratará de proponer trabajos posteriores que puedan continuar o mejorar los que se han obtenido en este trabajo.

7.1 Análisis del proyecto

Este trabajo tenía como objetivo principal la verificación de la seguridad del protocolo DP-3T ante varios ataques que vulneran la privacidad de los datos de los usuarios.

Tras la realización del proyecto, hemos conseguido realizar una primera definición del protocolo en la herramienta *Maude-NPA*, siendo esta la primera definición hecha para esta herramienta y además hemos conseguido definir una versión optimizada que tiene un mayor rendimiento y unos tiempos de ejecución más cortos.

Además, tras el análisis de los ataques que habíamos planteado en inicio, hemos podido determinar, que el protocolo *DP-3T* no es seguro ante ninguno de ellos, ya que gracias a la herramienta *DP-3T* hemos encontrado al menos una traza de ataque para cada uno de los ataques que hemos analizado.

7.2 Posibles mejoras y trabajos futuros

La mejora más natural sería analizar otros posibles ataques que puedan vulnerar la privacidad de los datos de los usuarios que utilizan el protocolo DP-3T, ya que hay otros que no se han analizado en este trabajo y que quedan pendientes de analizar.

Otro tipo de trabajo que se podría realizar sería analizar los datos obtenidos en este trabajo con los datos obtenidos con otras herramientas de verificación de protocolos, ya que hay más herramientas en el mercado, con la única intención de comparar resultados y métodos de trabajo, ya que aquí hemos comprobado que no es seguro.

Se podría desarrollar también un análisis comparativo entre los diferentes protocolos de rastreo por proximidad que existen, ya que aquí solamente hemos analizado el protocolo Dp-3T que es el que el Gobierno de España ha puesto al servicio de los ciudadanos, pero en otros países de Europa se usan aplicaciones con protocolos diferentes. También existe la posibilidad de analizar los protocolos que utilizan otro medio distinto al Bluetooth para la comunicación.

El campo de la criptografía es muy extenso, este tipo de protocolos son relativamente modernos y siguen evolucionando, por lo que la posibilidad de desarrollas nuevos trabajos de esta temática es alta.

8. Bibliografía

[1] Patel, Parth. «Is Your Telemedicine or Contact Tracing App Secure Enough?» Ryadel, 30 de mayo de 2020. https://www.ryadel.com/en/contact-tracing-app-privacy-encryption-security-risk-covid-19-telemedicine/

[2] Bay, Jason, Joel Kek, Alvin Tan, Chai Sheng Hau, Lai Yongquan, Janice Tan, y Tang Anh Quy. «BlueTrace: A Privacy-Preserving Protocol for Community-Driven Contact Tracing across Borders», s. f., 9.

[3] «Digital contact tracing - Wikipedia». Accedido 8 de julio de 2021. https://es.xcv.wiki/wiki/Digital_contact_tracing.

[4] Figliola, Patricia Moloney. «Digital Contact Tracing Technology: Overview and Considerations for Implementation», s. f., 3

[5] *DP-3T/documents*. Shell. 2020. Reprint, DP^3T, 2021. https://github.com/DP-3T/documents.

[6] . Escobar, Santiago, Catherine Meadows, y José Meseguer. «A Rewriting-Based Inference System for the NRL Protocol Analyzer and Its Meta-Logical Properties». *Theoretical Computer Science* 367, n.º 1-2 (noviembre de 2006): 162-202. https://doi.org/10.1016/j.tcs.2006.08.035.

[7] Vaudenay, Serge. «Analysis of DP3T», 2020. http://eprint.iacr.org/2020/399.

[8] «Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties». En *Foundations of Security Analysis and Design V*, editado por Alessandro Aldini, Gilles Barthe, y Roberto Gorrieri, 5705:1-50. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. https://doi.org/10.1007/978-3-642-03829-7_1.

9. Anexo 1: Resultado de la ejecución regular.

```
red summary(0,0) . --- States>> 1  Solutions>> 0
red summary(0,1) . --- States>> 4  Solutions>> 0
red summary(0,2) . --- States>> 7  Solutions>> 0
red summary(0,3) . --- States>> 12 Solutions>> 0
red summary(0,4) . --- States>> 12 Solutions>> 1
red initials(0,4) .
< 1 . 2 . 2 . 1 . 1 > (
:: nil ::
[ nil |
  -(u1 ; u2 ; eph(u1)),
  +(u2 ; u1 ; eph(u2)),
  -(s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
  +(s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: #0:Fresh ::
[ nil |
  +(u1 ; u2 ; eph(u1)),
  -(u2 ; u1 ; eph(u2)),
  +(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)), nil] )
|
(u1 ; u2 ; eph(u1)) !inI,
(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(u2 ; u1 ; eph(u2)) !inI,
(s ; sk(u1) ; t(u1, #0:Fresh)) !inI
|
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
-(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
+(s ; sk(u1) ; t(u1, #0:Fresh)),
-(s ; sk(u1) ; t(u1, #0:Fresh))
|
nil
```

10. Anexo 2: Resultado del Attack-State(1)

```
red summary(1,0) . --- States>> 1  Solutions>> 0
red summary(1,1) . --- States>> 2  Solutions>> 0
red summary(1,2) . --- States>> 4  Solutions>> 0
red summary(1,3) . --- States>> 5  Solutions>> 0
red summary(1,4) . --- States>> 7  Solutions>> 0
red summary(1,5) . --- States>> 9  Solutions>> 0
red summary(1,6) . --- States>> 15 Solutions>> 0
red summary(1,7) . --- States>> 15 Solutions>> 1
red initials(1,7) .
< 1 . 5 . 6 . 3 . 3 . 1 . 1 . 1 > (
:: nil ::
[ nil |
  -(s),
  -(sk(u1) ; t(u1, #0:Fresh)),
  +(s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(i),
  -(s ; sk(u1) ; t(u1, #0:Fresh)),
  +(i ; s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(u1 ; u2 ; eph(u1)),
  +(u2 ; u1 ; eph(u2)),
  -(i ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)),
  +(i ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(i ; s ; sk(u1) ; t(u1, #0:Fresh)),
  +(s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(i ; sk(u1) ; t(u1, #0:Fresh)),
  +(sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: #0:Fresh ::
[ nil |
  +(u1 ; u2 ; eph(u1)),
  -(u2 ; u1 ; eph(u2)),
  +(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)), nil] )
|
(u1 ; u2 ; eph(u1)) !inI,
(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(u2 ; u1 ; eph(u2)) !inI,
```

(s ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(i ; s ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(i ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(sk(u1) ; t(u1, #0:Fresh)) !inI
|
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)),
-(u1 ; i ; sk(u1) ; t(u1, #0:Fresh)),
+(i ; sk(u1) ; t(u1, #0:Fresh)),
-(i ; sk(u1) ; t(u1, #0:Fresh)),
+(sk(u1) ; t(u1, #0:Fresh)),
-(s),
-(sk(u1) ; t(u1, #0:Fresh)),
+(s ; sk(u1) ; t(u1, #0:Fresh)),
-(i),
-(s ; sk(u1) ; t(u1, #0:Fresh)),
+(i ; s ; sk(u1) ; t(u1, #0:Fresh)),
-(i ; s ; sk(u1) ; t(u1, #0:Fresh))
|
nil

11. Anexo 3: Resultado de Attack-State(2)

```
red summary(2,0) . --- States>> 1  Solutions>> 0
red summary(2,1) . --- States>> 3  Solutions>> 0
red summary(2,2) . --- States>> 5  Solutions>> 0
red summary(2,3) . --- States>> 5  Solutions>> 1
red initials(2,3) .
< 1 . 2 . 1 . 1 > (
:: nil ::
[ nil |
  -(i ; u2 ; eph(i)),
  +(u2 ; i ; eph(u2)),
  -(s ; sk(i) ; t(i, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(i ; s ; sk(i) ; t(i, #0:Fresh)),
  +(s ; sk(i) ; t(i, #0:Fresh)), nil]  &
:: #0:Fresh ::
[ nil |
  +(i ; u2 ; eph(i)),
  -(u2 ; i ; eph(u2)),
  +(i ; s ; sk(i) ; t(i, #0:Fresh)), nil] )
|
(u2 ; i ; eph(u2)) !inI,
(i ; u2 ; eph(i)) !inI,
(i ; s ; sk(i) ; t(i, #0:Fresh)) !inI
|
+(i ; u2 ; eph(i)),
generatedByIntruder(i ; u2 ; eph(i)),
-(i ; u2 ; eph(i)),
+(u2 ; i ; eph(u2)),
-(u2 ; i ; eph(u2)),
+(i ; s ; sk(i) ; t(i, #0:Fresh)),
-(i ; s ; sk(i) ; t(i, #0:Fresh))
|
nil
```

12. Anexo 4: Resultado de Attack-State(3)

```
red summary(3,0) . --- States>> 1  Solutions>> 0
red summary(3,1) . --- States>> 4  Solutions>> 0
red summary(3,2) . --- States>> 8  Solutions>> 0
red summary(3,3) . --- States>> 11 Solutions>> 0
red summary(3,4) . --- States>> 14  Solutions>> 0
red summary(3,5) . --- States>> 13  Solutions>> 1
red initials(3,5) .
< 1 . 6{1} . 2 . 2 . 1 . 1 > (
:: nil ::
[ nil |
  -(u1 ; u2 ; eph(u1)),
  +(u2 ; u1 ; eph(u2)),
  -(s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
  +(s ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: nil ::
[ nil |
  -(s ; sk(u1) ; t(u1, #0:Fresh)),
  +(sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: #0:Fresh ::
[ nil |
  +(u1 ; u2 ; eph(u1)),
  -(u2 ; u1 ; eph(u2)),
  +(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)), nil] )
|
(u1 ; u2 ; eph(u1)) !inI,
(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(u2 ; u1 ; eph(u2)) !inI,
(s ; sk(u1) ; t(u1, #0:Fresh)) !inI,
(sk(u1) ; t(u1, #0:Fresh)) !inI
|
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
-(u1 ; s ; sk(u1) ; t(u1, #0:Fresh)),
+(s ; sk(u1) ; t(u1, #0:Fresh)),
-(s ; sk(u1) ; t(u1, #0:Fresh)),
+(sk(u1) ; t(u1, #0:Fresh)),
-(s ; sk(u1) ; t(u1, #0:Fresh))
|
nil
```

13. Anexo 5: Resultado del Attack-State(4)

```
red summary(4,0) . --- States>> 1  Solutions>> 0
red summary(4,1) . --- States>> 2  Solutions>> 0
red summary(4,2) . --- States>> 3  Solutions>> 0
red summary(4,3) . --- States>> 2  Solutions>> 0
red summary(4,4) . --- States>> 3  Solutions>> 1
red initials(4,4) .
< 1 . 3 . 2 . 2 . 1 > (
:: nil ::
[ nil |
  -(eph(u1)),
  -(eph(i)),
  +(eph(u1) ; eph(i)), nil]  &
:: nil ::
[ nil |
  -(u1 ; i ; eph(u1)),
  +(i ; eph(u1)), nil]  &
:: nil ::
[ nil |
  -(i ; eph(u1)),
  +(eph(u1)), nil]  &
:: #0:Fresh ::
[ nil |
  +(u1 ; i ; eph(u1)),
  -(i ; u1 ; eph(i)),
  +(u1 ; i ; s ; sk(u1) ; t(u1, #0:Fresh)), nil] )
|
eph(u1) !inI,
eph(i) !inI,
(u1 ; i ; eph(u1)) !inI,
(i ; eph(u1)) !inI,
(i ; u1 ; eph(i)) !inI,
(eph(u1) ; eph(i)) !inI
|
+(u1 ; i ; eph(u1)),
-(u1 ; i ; eph(u1)),
+(i ; eph(u1)),
-(i ; eph(u1)),
+(eph(u1)),
generatedByIntruder(eph(i)),
-(eph(u1)),
-(eph(i)),
+(eph(u1) ; eph(i)),
generatedByIntruder(i ; u1 ; eph(i)),
-(i ; u1 ; eph(i))
|
nil
```

14. Anexo 6: Resultado del Attack-State(5)

```
red summary(5,0) . --- States>> 1  Solutions>> 0
red summary(5,1) . --- States>> 3  Solutions>> 0
red summary(5,2) . --- States>> 5  Solutions>> 0
red summary(5,3) . --- States>> 9  Solutions>> 0
red summary(5,4) . --- States>> 17 Solutions>> 0
red summary(5,5) . --- States>> 27 Solutions>> 0
red summary(5,6) . --- States>> 23 Solutions>> 0
red summary(5,7) . --- States>> 21 Solutions>> 2
red initials(5,7) .
(< 1 . 2 . 2 . 2 . 1 . 3 . 1 . 1 > (
:: nil ::
[ nil |
  -(i),
  -(u2 ; eph(u1)),
  +(i ; u2 ; eph(u1)), nil]  &
:: nil ::
[ nil |
  -(u1 ; eph(u2)),
  +(eph(u2)), nil]  &
:: nil ::
[ nil |
  -(u1 ; u2 ; eph(u1)),
  +(u2 ; eph(u1)), nil]  &
:: nil ::
[ nil |
  -(u2 ; u1 ; #1:EphimeralKey),
  +(u1 ; u2 ; eph(u1)), nil]  &
:: nil ::
[ nil |
  -(u2 ; u1 ; eph(u2)),
  +(u1 ; eph(u2)), nil]  &
:: nil ::
[ nil |
  -(i ; u2 ; eph(u1)),
  +(u2 ; u1 ; eph(u2)),
  -(s ; u2 ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: #0:Fresh ::
[ nil |
  +(u1 ; i ; eph(u1)),
  -(i ; u1 ; eph(i)),
  +(u1 ; i ; s ; sk(u1) ; t(u1, #0:Fresh)), nil] )
|
eph(u2) !inI,
(u1 ; eph(u2)) !inI,
(u1 ; u2 ; eph(u1)) !inI,
(u2 ; eph(u1)) !inI,
```

```
(u2 ; u1 ; #1:EphimeralKey) !inI,
(u2 ; u1 ; eph(u2)) !inI,
(i ; u2 ; eph(u1)) !inI
|
generatedByIntruder(u2 ; u1 ; #1:EphimeralKey),
-(u2 ; u1 ; #1:EphimeralKey),
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; eph(u1)),
-(i),
-(u2 ; eph(u1)),
+(i ; u2 ; eph(u1)),
-(i ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; eph(u2)),
-(u1 ; eph(u2)),
+(eph(u2)),
+(u1 ; i ; eph(u1))
|
nil)
< 1 . 2 . 2 . 2 . 1 . 3 . 1 . 2 > (
:: nil ::
[ nil |
  -(i),
  -(u2 ; eph(u1)),
  +(i ; u2 ; eph(u1)), nil]  &
:: nil ::
[ nil |
  -(u1 ; eph(u2)),
  +(eph(u2)), nil]  &
:: nil ::
[ nil |
  -(u1 ; u2 ; eph(u1)),
  +(u2 ; eph(u1)), nil]  &
:: nil ::
[ nil |
  -(u2 ; u1 ; eph(u2)),
  +(u1 ; eph(u2)), nil]  &
:: nil ::
[ nil |
  -(i ; u2 ; eph(u1)),
  +(u2 ; u1 ; eph(u2)),
  -(s ; u2 ; sk(u1) ; t(u1, #0:Fresh)), nil]  &
:: #1:Fresh ::
[ nil |
  +(u1 ; u2 ; eph(u1)), nil]  &
:: #0:Fresh ::
[ nil |
  +(u1 ; i ; eph(u1)),
  -(i ; u1 ; eph(i)),
  +(u1 ; i ; s ; sk(u1) ; t(u1, #0:Fresh)), nil] )
|
```

```
eph(u2) !inI,
(u1 ; eph(u2)) !inI,
(u1 ; u2 ; eph(u1)) !inI,
(u2 ; eph(u1)) !inI,
(u2 ; u1 ; eph(u2)) !inI,
(i ; u2 ; eph(u1)) !inI
|
+(u1 ; u2 ; eph(u1)),
-(u1 ; u2 ; eph(u1)),
+(u2 ; eph(u1)),
-(i),
-(u2 ; eph(u1)),
+(i ; u2 ; eph(u1)),
-(i ; u2 ; eph(u1)),
+(u2 ; u1 ; eph(u2)),
-(u2 ; u1 ; eph(u2)),
+(u1 ; eph(u2)),
-(u1 ; eph(u2)),
+(eph(u2)),
+(u1 ; i ; eph(u1))
|
nil
```